

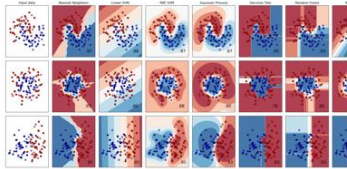
# 1. 机器学习简介

## Classification

Identifying which category an object belongs to.

**Applications:** Spam detection, image recognition.

**Algorithms:** SVM, nearest neighbors, random forest, and more...



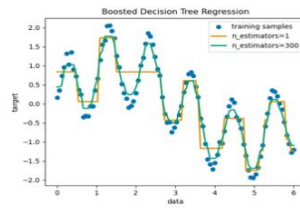
Examples

## Regression

Predicting a continuous-valued attribute associated with an object.

**Applications:** Drug response, Stock prices.

**Algorithms:** SVR, nearest neighbors, random forest, and more...



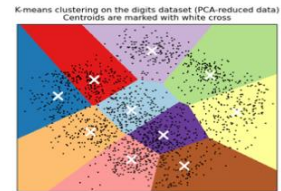
Examples

## Clustering

Automatic grouping of similar objects into sets.

**Applications:** Customer segmentation, Grouping experiment outcomes

**Algorithms:** k-Means, spectral clustering, mean-shift, and more...



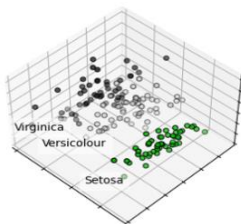
Examples

## Dimensionality reduction

Reducing the number of random variables to consider.

**Applications:** Visualization, Increased efficiency

**Algorithms:** PCA, feature selection, non-negative matrix factorization, and more...



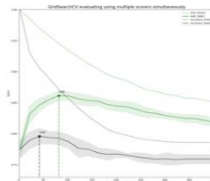
Examples

## Model selection

Comparing, validating and choosing parameters and models.

**Applications:** Improved accuracy via parameter tuning

**Algorithms:** grid search, cross validation, metrics, and more...



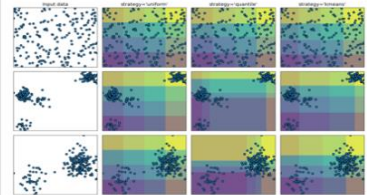
Examples

## Preprocessing

Feature extraction and normalization.

**Applications:** Transforming input data such as text for use with machine learning algorithms.

**Algorithms:** preprocessing, feature extraction, and more...

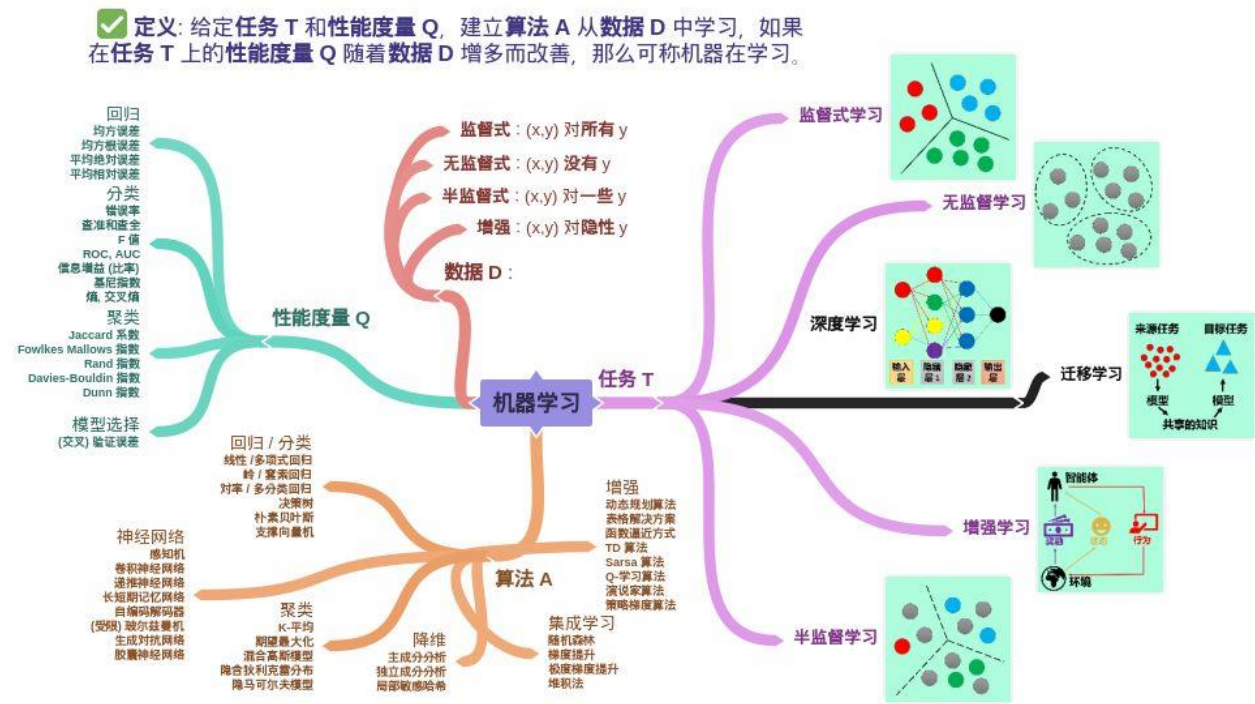


Examples

主页: <https://scikit-learn.org/stable/index.html>

api: <https://scikit-learn.org/stable/modules/classes.html>

1.1. 机器学习四要素



1.1.1. 数据 (Data)

计算机适合高效处理数值型结构化数据（表格化数据）

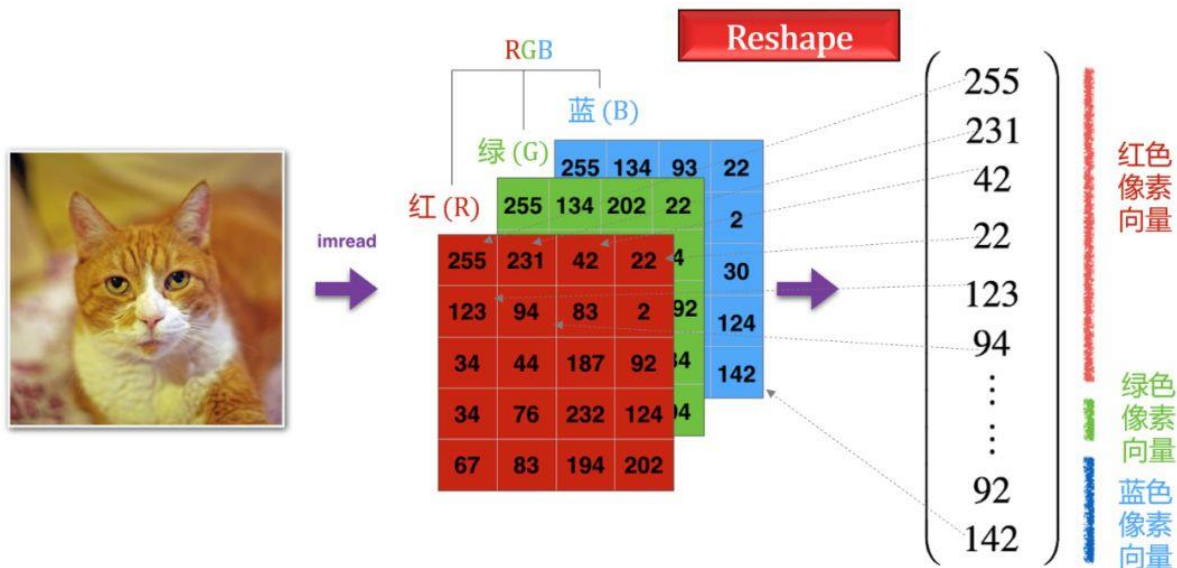
特征值 feature value		特征 feature	标签 label	
	得分	篮板	助攻	比赛结果
1	27	10	12	赢
2	33	9	9	输
3	51	10	8	输
4	40	13	15	赢

训练集  
training set

训练样例  
training example

原始数据分为非结构化数据（图像、文本）和结构化数据。数据加工的目的就是把非结构化数据转变成计算机能够处理的数值型变量。

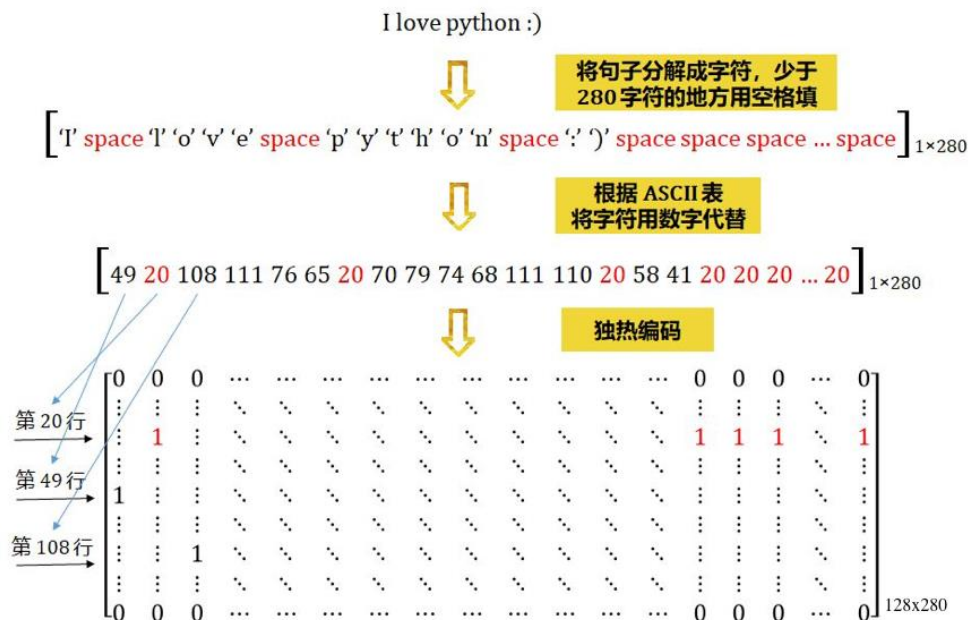
图像类数据



文本类数据

# ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]



## 数值型数据

	射门	传球	控球	比赛结果
1	9	42	12	赢
2	4	30	9	平
3	6	14	8	赢
4	0	22	15	输

$$\text{赢} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \text{平} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \text{输} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, y = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

## 1.1.2. 任务 (Task)

### 有监督学习 (supervised learning, 有标签)

利用输入数据及其对应标签来训练模型。在有监督学习中，**数据** = (**特征**，**标签**)，而其**主要任务**是**分类**和**回归**。如果预测的是离散值 (discrete value)，例如比赛结果赢或输，此类学习任务称为**分类** (classification)。如果预测的是连续值 (continuous value)，例如詹姆斯效率 65.1, 70.3 等等，此类学习任务称为**回归** (regression)。

### 无监督学习 (unsupervised learning, 无标签)

是找出输入数据的模式。在无监督学习中，**数据** = (**特征**，)。

**聚类** (clustering)，即将训练集中的数据分成若干组，每组成为一个簇(类别)，挖掘出数据之间潜在的关联。不同于分类，事先不确定类别个数。

**降维**，去除对学习作用不大或无作用的特征，降低数据的维度，帮助模型更好的理解数据，提高学习效率。

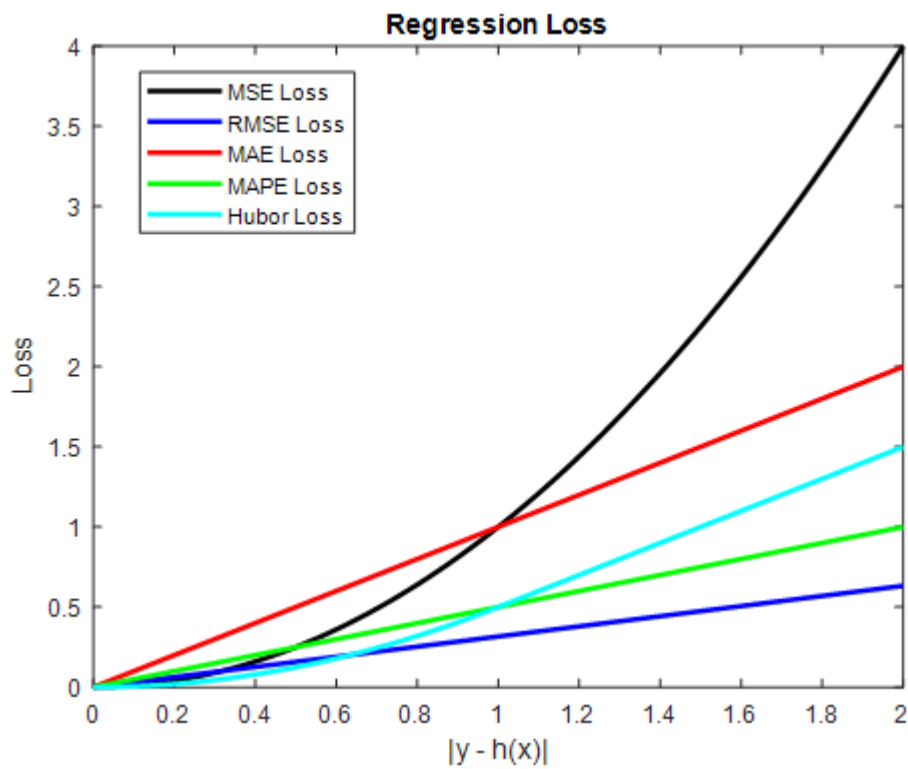


1.1.3. 性能度量 (Quality Metric)

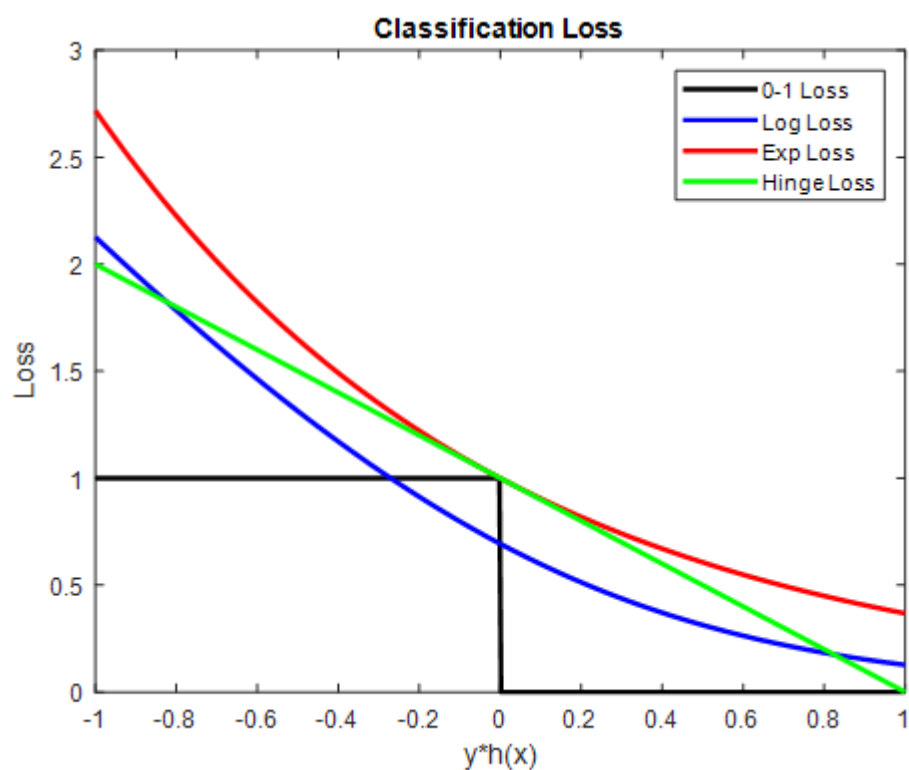
回归和分类任务中最常见的误差函数以及一些有用的性能度量如下。

任务	性能度量	
	误差函数	其它度量
回归	均方、均方根、平均绝对/相对误差、Huber	-
分类	0-1、对数、指数、合页	错误率、查全率、查准率、 $F_1$ 得分、AUC、ROC

误差函数类型	函数表法式 $E_D[h]$
均方误差 (mean square error, MSE)	$\frac{1}{m} \sum_{i=1}^m [h(x^{(i)}) - y^{(i)}]^2$
均方根误差: 均方误差的平方根 (root mean square error, RMSE)	$\sqrt{\frac{1}{m} \sum_{i=1}^m [h(x^{(i)}) - y^{(i)}]^2}$
平均绝对误差: 绝对误差的平均值 (mean absolute error, MAE)	$\frac{1}{m} \sum_{i=1}^m  h(x^{(i)}) - y^{(i)} $
平均相对误差: 相对误差的平均值 (mean absolute percentage error, MAPE)	$\frac{1}{m} \sum_{i=1}^m \left  \frac{h(x^{(i)}) - y^{(i)}}{y^{(i)}} \right $
Huber	$\frac{1}{m} \sum_{i=1}^m \begin{cases} [h(x^{(i)}) - y^{(i)}]^2, &  h(x^{(i)}) - y^{(i)}  \leq \delta \\ 2\delta( h(x^{(i)}) - y^{(i)}  - \delta), &  h(x^{(i)}) - y^{(i)}  > \delta \end{cases}$

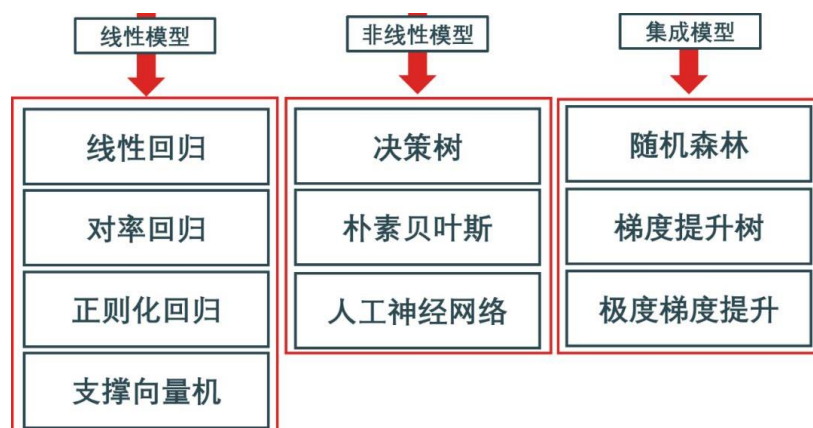


误差函数类型	函数表法式 $E_D[h]$
<b>0-1</b> (misclassification)	$\frac{1}{m} \sum_{i=1}^m \begin{cases} 1, & y^{(i)} \cdot h(x^{(i)}) < 0 \\ 0, & y^{(i)} \cdot h(x^{(i)}) > 0 \end{cases} = I\{y^{(i)} \cdot h(x^{(i)}) < 0\}$
对数 (logarithm)	$\frac{1}{m} \sum_{i=1}^m \ln(1 + \exp(-2y^{(i)} \cdot h(x^{(i)})))$
指数 (exponential)	$\frac{1}{m} \sum_{i=1}^m \exp(-y^{(i)} \cdot h(x^{(i)}))$
合页 (hinge)	$\frac{1}{m} \sum_{i=1}^m (1 - y^{(i)} \cdot h(x^{(i)}))^+$



### 1.1.4. 模型 (Model)

有监督模型



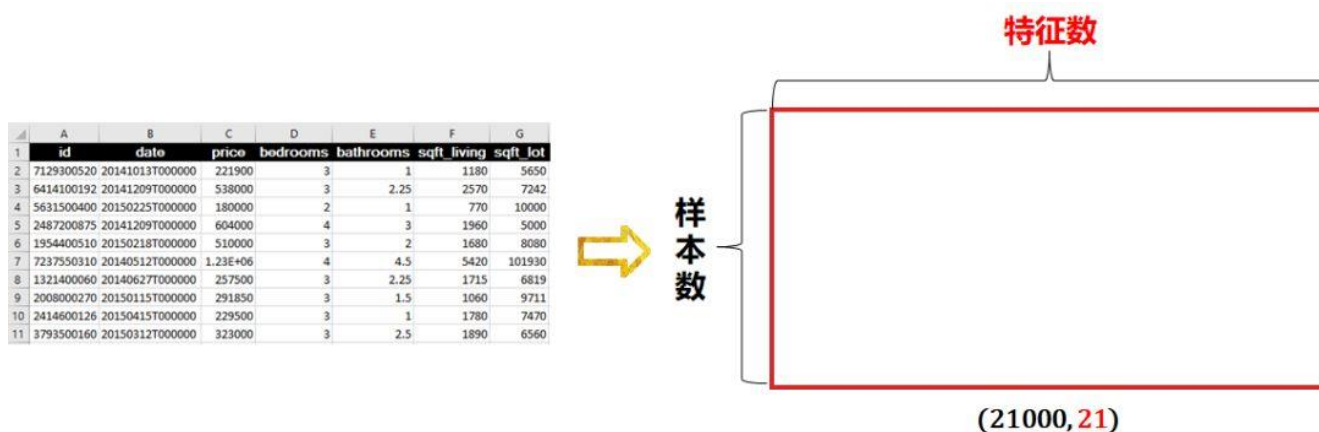
无监督模型包括各种聚类分析 (KMeans, DBSCAN)、主成分分析 (PCA)、独立成分分析 (ICA)、隐含狄利克雷分配 (LDA) 等等。

## 1.2. 数据模块

### 1.2.1. 数据格式

Numpy 二维数组 (ndarray) / Pandas DataFrame 的稠密数据 (dense data), 通常都是这种格式。

SciPy 矩阵 (scipy.sparse.matrix) 的稀疏数据 (sparse data), 比如文本分析每个单词 (字典有 100000 个词) 做独热编码得到矩阵有很多 0, 这时用 ndarray 就不合适了, 太耗内存。



### 1.2.2. 自带数据集

打包好的数据: 对于小数据集, 用 `sklearn.datasets.load_*`

分流下载数据: 对于大数据集, 用 `sklearn.datasets.fetch_*`

随机创建数据: 为了快速展示, 用 `sklearn.datasets.make_*`

示例: 鸢尾花, 包括 150 条鸢尾花的四个特征 (萼片长/宽和花瓣长/宽) 和三个类别。



## 1.3. 核心 API

### 1.3.1. 估计器

定义: 任何可以基于数据集对一些参数进行估计的对象都被称为估计器。

两个核心点：1. 需要输入数据，2. 可以估计参数。估计器首先被创建，然后被拟合。

**创建估计器：**需要设置一组超参数，比如

- 线性回归里超参数 `fit_intercept=True`
- K 均值里超参数 `n_clusters=3`

**拟合估计器：**需要训练集。

在有监督学习中的代码范式为

```
model.fit( X_train, y_train )
```

在无监督学习中的代码范式为

```
model.fit( X_train )
```

拟合之后可以访问 `model` 里学到的参数，比如线性回归里的特征前的系数 `coef_`，或 K 均值里聚类标签 `label`。

```
model.coef_
```

```
model.label
```

## ➤ 线性回归

首先从 `sklearn` 下的 `linear_model` 中引入 `LinearRegression`，再创建估计器起名 `model`，设置超参数 `normalize` 为 `True`，指的在每个特征值上做标准化，这样会加速数值运算。

```
from sklearn.linear_model import LinearRegression
model = LinearRegression(normalize=True)
model

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None)

x = np.arange(10)
y = 2*x+1
X = x[:, np.newaxis]
# 创建一个简单数据集
model.fit(X,y)

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None)

print(model.coef_)
print(model.intercept_)
# 斜率为 2，截距为 1
```



## ➤ K 均值

首先从 sklearn 下的 cluster 中引入 KMeans，再创建估计器起名 model，设置超参数 n\_cluster 为 3。

```
from sklearn.cluster import KMeans
```

```
model = KMeans(n_clusters=3)
```

```
model
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

```
X = iris.data[:,0:2] #只取两个特征 (萼片长、萼片宽)
```

```
model.fit(X)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

```
print( model.cluster_centers_) #簇中心
```

```
print( model.labels_) #聚类后的标签
```

```
print( model.inertia_) #所有点到对应的簇中心的距离平方和 (越小越好)
```

```
print( iris.target ) #需要强调的是真实标签 iris.label 和聚类标签 model.labels_ 看起来差的很远。
```

## ➤ 小节

```
# 有监督学习
```

```
from sklearn.xxx import SomeModel
```

```
# xxx 可以是 linear_model 或 ensemble 等
```

```
model = SomeModel( hyperparameter )
```

```
model.fit( X, y )
```

```
# 无监督学习
```

```
from sklearn.xxx import SomeModel
```

```
# xxx 可以是 cluster 或 decomposition 等
```

```
model = SomeModel( hyperparameter )
```

```
model.fit( X )
```

## 1.3.2. 预测器

定义：预测器在估计器上做了一个延展，延展出预测的功能。

两个核心点：1. 基于学到的参数预测，2. 预测有很多指标。最常见的就是 `predict()` 函数：

```
model.predict(X_test): 评估模型在新数据上的表现
model.predict(X_train): 确认模型在老数据上的表现
```

首先将数据分成 80:20 的训练集 (`X_train, y_train`) 和测试集 (`X_test, y_test`)，用从训练集上拟合 `fit()` 的模型 `model` 在测试集上预测 `predict()`。

```
from sklearn.datasets import load_iris
iris = load_iris()

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test
= train_test_split( iris['data'], iris['target'], test_size=0.2 )
```

### ➤ 逻辑回归

首先从 `sklearn` 下的 `linear_model` 中引入 `LogisticRegression`，再创建估计器 `model`，设置超参数 `multi_class` 为 `multinomial` 因为有三种鸢尾花，是个多分类问题。

接着再训练集上拟合参数，这时估计器 `model` 里面已经可以访问这些参数了。

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression (solver='lbfgs', multi_class='multinomial')

model.fit(X_train, y_train)

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, max_iter=100, multi_class='multinomial',
                    n_jobs=None, penalty='l2', random_state=None, solver='lbfgs',
                    tol=0.0001, verbose=0, warm_start=False)
```

### **predict & predict\_proba**

对于分类问题，预测类别用 `predict()`，预测该类别的可能性用 `predict_proba()`。

```
y_pred = model.predict( X_test )
p_pred = model.predict_proba( X_test )
```

### **score & decision\_function**

预测器里还有额外的两个函数可以使用。在分类问题中

score() 返回的是分类准确率

decision\_function() 返回的是每个样例在每个类下的置信分数值

## ➤ K 均值

```
from sklearn.cluster import KMeans
```

```
model = KMeans(n_clusters=3)
```

```
model.fit(X_train[:, 0:2])
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

```
idx_pred = model.predict(X_test[:, 0:2])
```

```
idx_score = model.score(X_test[:, 0:2])
```

估计器都有 fit() 方法，预测器都有 predict() 和 score() 方法，但不是每个预测器都有 predict\_proba() 和 decision\_function() 方法

# 有监督学习

```
from sklearn.xxx import SomeModel
```

# xxx 可以是 linear\_model 或 ensemble 等

```
model = SomeModel(hyperparameter)
```

```
model.fit(X, y)
```

# 无监督学习

```
from sklearn.xxx import SomeModel
```

# xxx 可以是 cluster 或 decomposition 等

```
model = SomeModel(hyperparameter)
```

```
model.fit(X)
```

## ➤ 小节

### 1.3.3. 转换器

定义：转换器也是一种估计器，两者都带拟合功能，但估计器做完拟合用于预测，而转换器做完拟合用于转换。

核心点：估计器里 fit + predict，转换器里 fit + transform。

## ➤ 分类型变量编码

将分类型变量 (categorical) 编码成数值型变量 (numerical)

### LabelEncoder & OrdinalEncoder

LabelEncoder 和 OrdinalEncoder 都可以将字符转成数字，但是

LabelEncoder 的输入是一维，比如 1d ndarray

OrdinalEncoder 的输入是二维，比如 DataFrame

```
enc = ['win','draw','lose','win'] #编码列表
dec = ['draw','draw','win']      #解码列表
```

```
from sklearn.preprocessing import LabelEncoder
LE = LabelEncoder()
print( LE.fit(enc) )
print( LE.classes_ )
print( LE.transform(dec) )

LabelEncoder()
['draw' 'lose' 'win']
[0 0 2]
```

```
from sklearn.preprocessing import OrdinalEncoder
OE = OrdinalEncoder()
enc_DF = pd.DataFrame(enc)
dec_DF = pd.DataFrame(dec)
print( OE.fit(enc_DF) )
print( OE.categories_ )
print( OE.transform(dec_DF) )

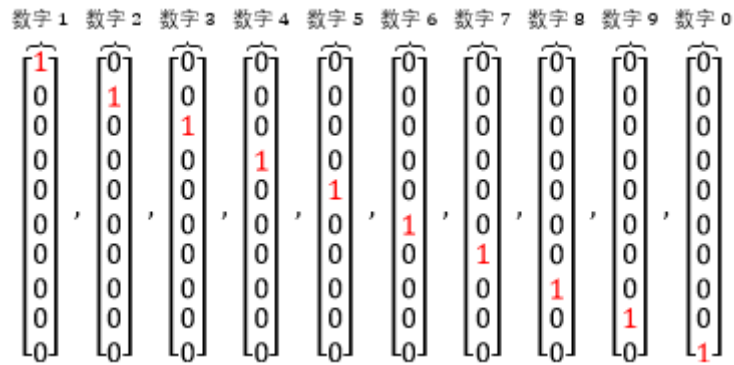
OrdinalEncoder(categories='auto', dtype=<class 'numpy.float64'>)
[array(['draw', 'lose', 'win'], dtype=object)]
[[0.]
 [0.]
 [2.]]
```

机器学习算法会认为两个临近的值比两个疏远的值要更相似。显然这样不对（比如，0 和 1 比 0 和 2 距离更近，难道 draw 和 win 比 draw 和 lose 更相似？）

### OneHotEncoder



独热编码其实就是把一个整数用向量的形式表现。下图就是对数字 0-9 做独热编码。



转换器 `OneHotEncoder` 可以接受两种类型的输入：

- A. 用 `LabelEncoder` 编码好的一维数组
- B. `DataFrame`

用 `LabelEncoder` 编码好的一维数组（元素为整数），重塑（用 `reshape(-1,1)`）成二维数组作为 `OneHotEncoder` 输入。

```
from sklearn.preprocessing import OneHotEncoder

OHE = OneHotEncoder()

num = LE.fit_transform( enc )

print( num )

OHE_y = OHE.fit_transform( num.reshape(-1,1) ) # [[2],[0],[1],[2]]

OHE_y

OHE_y.toarray()

[[2 0 1 2]]

<4x3 sparse matrix of type
'<class 'numpy.float64'>'
with 4 stored elements
in Compressed Sparse Row format>

array([[0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

用 `DataFrame` 作为 `OneHotEncoder` 输入。

```
OHE = OneHotEncoder()
```

```
OHE.fit_transform( enc_DF ).toarray()
```

```
array([[0., 0., 1.],  
       [1., 0., 0.],  
       [0., 1., 0.],  
       [0., 0., 1.]])
```

## ➤ 特征缩放

规范化 (normalize) 或标准化 (standardize) 数值型变量

数据要做的最重要的转换之一是特征缩放 (feature scaling)。当输入的数值的量纲不同时，机器学习算法的性能都不会好。

具体来说，对于某个特征，我们有两种方法：

- A. 标准化 (standardization)：每个维度的特征减去该特征均值，除以该维度的标准差。
- B. 规范化 (normalization)：每个维度的特征减去该特征最小值，除以该特征的最大值与最小值之差。

---

规范化  $z = \frac{x - \text{最小值}}{\text{最大值} - \text{最小值}}$ ：将  $z$  缩放到 0 和 1 之间，用 sklearn 里面 MinMaxScaler 函数

标准化  $z = \frac{x - \text{均值}}{\text{标准差}}$ ：将  $z$  缩放到以 0 为中心而分散为 1 的区间，用 sklearn 里面 StandardScaler 函数

---

### MinMaxScaler

```
from sklearn.preprocessing import MinMaxScaler  
  
X = np.array( [0, 0.5, 1, 1.5, 2, 100] )  
  
X_scale = MinMaxScaler().fit_transform( X.reshape(-1,1) )  
  
X_scale  
  
array([[0. ],  
       [0.005],  
       [0.01 ],  
       [0.015],  
       [0.02 ],  
       [1. ]])
```

### StandardScaler

```
from sklearn.preprocessing import StandardScaler

X_scale = StandardScaler().fit_transform( X.reshape(-1,1) )

X_scale

array([[ -0.47424487],
       [ -0.46069502],
       [ -0.44714517],
       [ -0.43359531],
       [ -0.42004546],
       [  2.23572584]])
```

**警告：** fit() 函数只能作用在训练集上，千万不要作用在测试集上，不然就犯了数据窥探的错误！拿标准化举例，用训练集 fit 出来的均值和标准差参数，来对测试集做标准化。

## 1.4. 高级 API

元估计器 (meta-estimator)，即由很多基估计器 (base estimator) 组合成的估计器。元估计器把估计器当成参数。

五大元估计器如下：

```
#模型集成
ensemble.BaggingClassifier
ensemble.VotingClassifier
#多分类和多标签
multiclass.OneVsOneClassifier
multiclass.OneVsRestClassifier
#多输出
multioutput.MultiOutputClassifier
#模型选择
model_selection.GridSearchCV
model_selection.RandomizedSearchCV
#流水线
pipeline.Pipeline
```

### 1.4.1. Ensemble 估计器

Ensemble 估计器是用来做集成学习，该估计器里面有若干个分类器 (classifier) 或回归器 (regressor)。

分类器统计每个子分类器的预测类别数，再用「多数投票」原则得到最终预测。

回归器计算每个子回归器的预测平均值。

**AdaBoostClassifier**: 逐步提升分类器

**AdaBoostRegressor**: 逐步提升回归器

**BaggingClassifier**: 装袋分类器

**BaggingRegressor**: 装袋回归器

**GradientBoostingClassifier**: 梯度提升分类器

**GradientBoostingRegressor**: 梯度提升回归器

**RandomForestClassifier**: 随机森林分类器

**RandomForestRegressor**: 随机森林回归器

**VotingClassifier**: 投票分类器

**VotingRegressor**: 投票回归器

最常用的 Ensemble 估计器排列如下：

基于鸢尾花数据 `iris`，拿含 **同质** 估计器 `RandomForestClassifier` 和含 **异质** 估计器 `VotingClassifier` 来举例。首先将数据分成 80:20 的训练集和测试集，并引入 `metrics` 来计算各种性能指标。

```
from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn import metrics

iris = load_iris()

X_train, X_test, y_train, y_test = train_test_split(iris['data'], iris['target'], test_size=0.2)
```

## **RandomForestClassifier**

随机森林 (random forest) 是决策树 (decision tree) 的一种集成模型，每棵决策树处理的数据用装袋法 (bagging) 生成。随机森林可以减小预测的方差，并且可以评估特征重要性。

```
from sklearn.ensemble import RandomForestClassifier

RF = RandomForestClassifier(n_estimators=4, max_depth=5)

RF.fit(X_train, y_train)

RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                        max_depth=5, max_features='auto', max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=4, n_jobs=None,
                        oob_score=False, random_state=None, verbose=0,
                        warm_start=False)
```



```
print( RF.n_estimators )
```

```
RF.estimators_
```

```
4
```

```
[DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
    max_features='auto', max_leaf_nodes=None,
    min_impurity_decrease=0.0, min_impurity_split=None,
    min_samples_leaf=1, min_samples_split=2,
    min_weight_fraction_leaf=0.0, presort=False,
    random_state=1264864033, splitter='best'),
    DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
    max_features='auto', max_leaf_nodes=None,
    min_impurity_decrease=0.0, min_impurity_split=None,
    min_samples_leaf=1, min_samples_split=2,
    min_weight_fraction_leaf=0.0, presort=False,
    random_state=202883886, splitter='best'),
    DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
    max_features='auto', max_leaf_nodes=None,
    min_impurity_decrease=0.0, min_impurity_split=None,
    min_samples_leaf=1, min_samples_split=2,
    min_weight_fraction_leaf=0.0, presort=False,
    random_state=1981146806, splitter='best'),
    DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
    max_features='auto', max_leaf_nodes=None,
    min_impurity_decrease=0.0, min_impurity_split=None,
    min_samples_leaf=1, min_samples_split=2,
    min_weight_fraction_leaf=0.0, presort=False,
    random_state=783144523, splitter='best')]
```

```
print ("RF - Accuracy (Train): %.4g" % metrics.accuracy_score(y_train, F.predict(X_train)) )
```

```
print ("RF - Accuracy (Test): %.4g" % metrics.accuracy_score(y_test, RF.predict(X_test)) )
```

```
RF - Accuracy (Train): 0.9833
```

```
RF - Accuracy (Test): 1
```

## VotingClassifier

和随机森林由同质分类器「决策树」不同，投票分类器由若干个异质分类器组成。下例用 VotingClassifier 建立个含有对率回归 (LR)、随机森林 (RF) 和高斯朴素贝叶斯 (GNB) 三个分类器的集成模型。

```

from sklearn.linear_model import LogisticRegression

from sklearn.naive_bayes import GaussianNB

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import VotingClassifier

```

```
LR = LogisticRegression( solver='lbfgs', multi_class='multinomial')
```

```
RF = RandomForestClassifier(n_estimators=5 )
```

```
GNB = GaussianNB()
```

```
Ensemble = VotingClassifier(estimators=[('lr', LR), ('nf', RF), ('gnb', GNB)], voting='hard')
```

```
Ensemble.fit(X_train, y_train )
```

```
VotingClassifier(estimators=[('lr', LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, max_iter=100, multi_class='multinomial',
n_jobs=None, penalty='l2', random_state=None, solver='lbfgs',
tol=0.0001, verbose=0, warm_start=False)), ('rf', ...e, verbose=0,
warm_start=False)), ('gnb', GaussianNB(priors=None, var_smoothing=1e-09))],
flatten_transform=None, n_jobs=None, voting='hard', weights=None)
```

```
print(len(Ensemble.estimators_))
```

```
Ensemble.estimators_
```

```
3
[LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, max_iter=100, multi_class='multinomial',
n_jobs=None, penalty='l2', random_state=None, solver='lbfgs',
tol=0.0001, verbose=0, warm_start=False),
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
max_depth=None, max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=5, n_jobs=None,
oob_score=False, random_state=None, verbose=0,
warm_start=False),
GaussianNB(priors=None, var_smoothing=1e-09)]
```

```
LR.fit( X_train, y_train )
```

```
RF.fit( X_train, y_train )
```

```
GNB.fit( X_train, y_train )
```

```
print("LR-Accuracy(Train):%.4g" % metrics.accuracy_score(y_train,LR.predict(X_train)))
```

```
print("RF-Accuracy(Train):%.4g" % metrics.accuracy_score(y_train, RF.predict(X_train)))
```

```
print("GNB-Accuracy(Train):%.4g" % metrics.accuracy_score(y_train, GNB.predict(X_train)))
```

```
print("Ensemble-Accuracy(Train):%.4g" %
metrics.accuracy_score(y_train,Ensemble.predict(X_train)))
```

```
print("LR-Accuracy(Test):%.4g" % metrics.accuracy_score(y_test,LR.predict(X_test)))
```

```
print("RF-Accuracy(Test):%.4g" % metrics.accuracy_score(y_test, RF.predict(X_test)))
```

```
print("GNB-Accuracy(Test):%.4g" % metrics.accuracy_score(y_test, GNB.predict(X_test)))

print("Ensemble-Accuracy(Test):%.4g" %
      metrics.accuracy_score(y_test, Ensemble.predict(X_test)))
```

```
LR - Accuracy (Train): 0.975
RF - Accuracy (Train): 0.9833
GNB - Accuracy (Train): 0.95
Ensemble - Accuracy (Train): 0.9833
```

```
LR - Accuracy (Test): 1
RF - Accuracy (Test): 1
GNB - Accuracy (Test): 1
Ensemble - Accuracy (Test): 1
```

### 1.4.2. Multiclass 估计器

`sklearn.multiclass` 可以处理多类别 (multi-class) 和多标签 (multi-label) 的分类问题。基于数字数据集 `digits`，将数据分成 80:20 的训练集和测试集。

```
from sklearn.datasets import load_digits

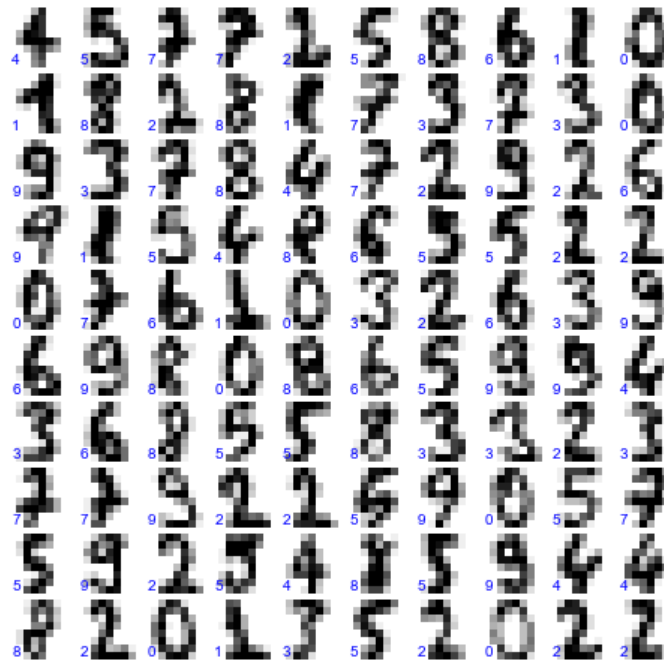
digits = load_digits()

digits.keys()

dict_keys(['data', 'target', 'target_names', 'images', 'DESCR'])

X_train, X_test, y_train, y_test = train_test_split(digits['data'], digits['target'], test_size=0.2)
```

训练集和测试集分别有 1437 和 360 张图像。每张照片是包含  $8 \times 8$  的像素，将其打平 (flatten) 把 2 维的  $8 \times 8$  重塑成 1 维的 64。



## ➤ 多类别分类

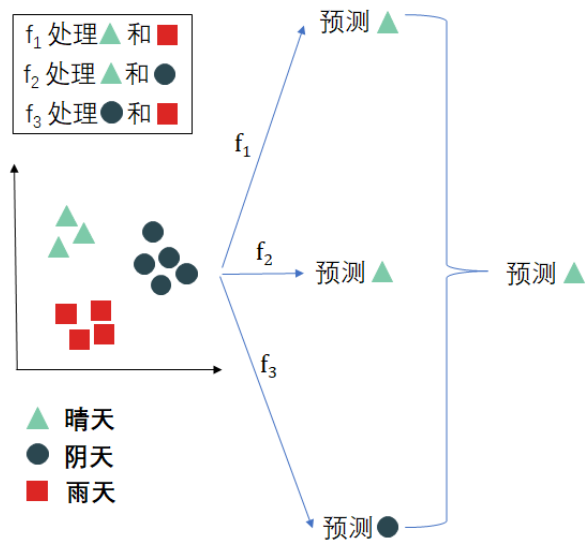
两种分类策略

- 一对一(One vs One, OvO): 一个分类器用来处理数字 0 和数字 1, 一个用来处理数字 0 和数字 2, 一个用来处理数字 1 和 2, 以此类推。N 个类需要  $N(N-1)/2$  个分类器。
- 一对其他(One vs All, OvA): 训练 10 个二分类器, 每一个对应一个数字, 第一个分类 1 和「非 1」, 第二个分类 2 和「非 2」, 以此类推。N 个类需要 N 个分类器。

### OneVsOneClassifier

考虑一个具体天气多分类问题, 天气可以是晴天、阴天和雨天, 在 OvO 中, 三个分类器为 f1, f2 和 f3。





```
from sklearn.multiclass import OneVsOneClassifier
from sklearn.linear_model import LogisticRegression

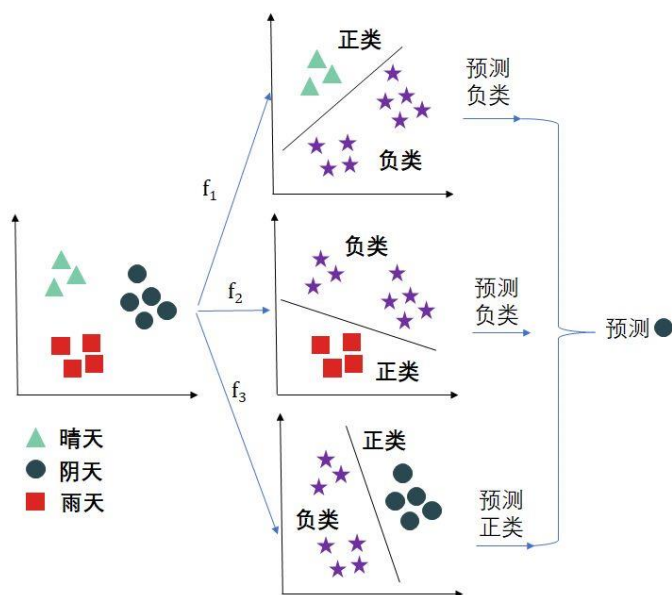
ovo_lr = OneVsOneClassifier(LogisticRegression(solver='lbfgs', max_iter=200))
ovo_lr.fit(X_train, y_train)
print(len(ovo_lr.estimators_)) #45
print("OvO LR-Accuracy(Train):%.4g%" % metrics.accuracy_score(y_train, ovo_lr.predict(X_train)))
print("OVO LR-Accuracy(Test):%.4g%" % metrics.accuracy_score(y_test, ovo_lr.predict(X_test)))
```

OvO LR - Accuracy (Train): 1

OvO LR - Accuracy (Test): 0.9806

## OneVsRestClassifier

在 OvA 中，把数据分成“某个”和“其他”。三分类分解成三个二分类，对应的分类器为  $f_1$ ,  $f_2$  和  $f_3$ 。



```

from sklearn.multiclass import OneVsRestClassifier

ova_lr = OneVsRestClassifier(LogisticRegression(solver='lbfgs', max_iter=800))

ova_lr.fit(X_train, y_train)

print(len(ova_lr.estimators_)) # 10

print("OvA LR-Accuracy(Train):%.4g%" % metrics.accuracy_score(y_train, ova_lr.predict(X_train)))

print("OVA LR-Accuracy(Test):%.4g%" % metrics.accuracy_score(y_test, ova_lr.predict(X_test)))

OvA LR - Accuracy (Train): 0.9993

OvA LR - Accuracy (Test): 0.9639

```

### ➤ 多标签分类

在手写数字的例子，为每个数字设计了多标签：

- 标签 1 - 奇数、偶数
- 标签 2 - 小于等于 4，大于 4

```

from sklearn.multiclass import OneVsRestClassifier

# OneVsRestClassifier 也可以用来做多标签分类

y_train_multilabel = np.c_[y_train%2==0, y_train<=4]

ova_ml = OneVsRestClassifier(LogisticRegression(solver='lbfgs', max_iter=800))

ova_ml.fit(X_train, y_train_multilabel)

print(len(ova_ml.estimators_)) # 2

```

### 1.4.3. Multioutput 估计器

sklearn.multioutput 可以处理多输出 (multi-output) 的分类问题。

多输出分类是多标签分类的泛化，在这里每一个标签可以是多类别 (大于两个类别) 的。

Multioutput 估计器有两个：

- MultiOutputRegressor: 多输出回归
- MultiOutputClassifier: 多输出分类

#### MultiOutputClassifier

在手写数字的例子，我们也为特意每个数字设计了多标签而且每个标签的类别都大于二。

- 标签 1 - 小于等于 4，4 和 7 之间，大于等于 7 (三类)

- 标签 2 - 数字本身 (十类)

```
from sklearn.multioutput import MultiOutputClassifier
from sklearn.ensemble import RandomForestClassifier

y_train_1st = y_train.copy()
y_train_1st[y_train<=4] = 0
y_train_1st[np.logical_and(y_train>4,y_train<7)] = 1
y_train_1st[y_train>=7] = 2

y_train_multioutput = np.c_[y_train_1st, y_train]

MO = MultiOutputClassifier(RandomForestClassifier(n_estimators=100))

MO.fit(X_train,y_train_multioutput)

MO.predict(X_test[:5,:])
```

#### 1.4.4. Model Selection 估计器

模型选择 (Model Selction) 在机器学习非常重要，它主要用于评估模型表现，常见的 Model Selection 估计器有以下几个：

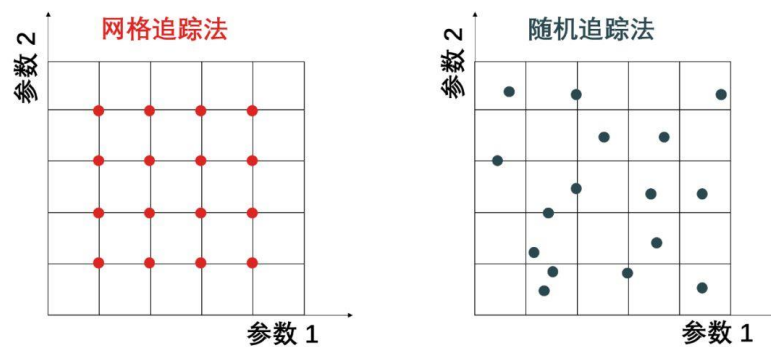
- cross\_validate: 评估交叉验证的表现。
- learning\_curve: 建立学习曲线。
- GridSearchCV: 用交叉验证从网格中一组超参数搜索出最佳超参数。
- RandomizedSearchCV: 用交叉验证从一组随机超参数搜索出最佳超参数。

##### ➤ 交叉验证

K-折交叉验证集 (K-fold cross validation set)，就是把整个数据集平均但随机分成 K 份，每份大概包含  $m/K$  个数据 (m 是总数据数)。在这 K 份，每次选 1 份作为训练集在拟合参数  $w_\lambda$ ，把参数用在剩下 K-1 份验证集上计算误差。由于遍历了这 K 份数据，因此该操作称为交叉验证。



调参的估计器，网格追踪(GridSearchCV)和随机追踪(RandomizedSearchCV)





```

from time import time
from scipy.stats import randint
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier

X, y=digits.data, digits.target
RFC = RandomForestClassifier(n_estimators=20)

# Randomized Search
param_dist = {"max_depth": [3, 5], "max_features": randint(1, 11),
              "min_samples_split": randint(2, 11), "criterion":["gini","entropy"]}
n_iter_search = 20
random_search = RandomizedSearchCV( RFC, param_distributions=param_dist,
                                   n_iter=n_iter_search, cv=5)

start=time()
random_search.fit(X, y)
print("RandomizedSearchCV took %.2f seconds for %d candidates parameter settings."
      %((time()- start), n_iter_search))

print( random_search.best_params_)
print( random_search.best_score_)

# Grid Search
param_grid = {"max_depth": [3, 5], "max_features": [1, 3, 10],
              "min_samples_split": [2, 3, 10], "criterion":["gini","entropy"]}
grid_search = GridSearchCV(RF, param_grid=param_grid, cv=5)
start=time()
grid_search.fit(X, y)
print("\nGridSearchCV took %.2f seconds for %d candidate parameter settings."
      %(time()-start, len(grid_search.cv_results_['params'])))
print( grid_search.best_params_)
print( grid_search.best_score )

```

### 1.4.5. Pipeline 估计器

Pipeline 估计器又叫流水线，把各种估计器串联 (Pipeline)的方式组合，提高效率。

#### ➤ Pipeline

Pipeline 将若干个估计器按顺序连在一起，比如：特征提取 -> 降维 -> 拟合 -> 预测  
在整个 Pipeline 中，它的属性永远和最后一个估计器属性一样。

- 如果最后一个估计器是预测器，那么 Pipeline 是预测器
- 如果最后一个估计器是转换器，那么 Pipeline 是转换器

例：先填补缺失值-再标准化

首先引入 Pipeline，再引入处理缺失值的转换器 SimpleImputer 和规范化的转换器

```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler

X= np.array([[50,40,30,5,7,10,9,np.NaN,12],
             [1.68,1.83,1.77,np.NaN,1.9,1.65,1.88,np.NaN,1.75]])

X = np.transpose(x)

pipe = Pipeline([("impute", SimpleImputer(missing_values=np.nan, strategy='mean')),
                 ('normalize', MinMaxScaler())])

X_proc = pipe.fit_transform(X)

X_impute = SimpleImputer(missing_values=np.nan, strategy='mean').fit_transform(X)
X_normalize = MinMaxScaler().fit_transform(X_impute)
```

## 2. 回归算法与应用

### 2.1. 线性回归

API:

```
sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0,
fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs',
max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)
```

Reference:

[https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.LogisticRegression.html#sklearn.linear\\_model.LogisticRegression](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression)

### 2.2. Lasso 回归

API:

```
sklearn.linear_model.Lasso(alpha=1.0, *, fit_intercept=True, precompute=False, copy_X=True,
max_iter=1000, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic')
```

Reference:

[https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.Lasso.html#sklearn.linear\\_model.Lasso](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html#sklearn.linear_model.Lasso)

## 2.3. Ridge 岭回归

API:

```
sklearn.linear_model.Ridge(alpha=1.0, *, fit_intercept=True, copy_X=True, max_iter=None, tol=0.0001, solver='auto', positive=False, random_state=None)
```

Reference:

[https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.Ridge.html#sklearn.linear\\_model.Ridge](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html#sklearn.linear_model.Ridge)

## 2.4. ElasticNet 弹性网

API:

```
sklearn.linear_model.ElasticNet(alpha=1.0, *, l1_ratio=0.5, fit_intercept=True, precompute=False, max_iter=1000, copy_X=True, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic')
```

Reference:

[https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.ElasticNet.html#sklearn.linear\\_model.ElasticNet](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html#sklearn.linear_model.ElasticNet)

# 3. 分类算法与应用

## 3.1. 逻辑回归

API:

```
sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)
```

Reference:

[https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.LogisticRegression.html#sklearn.linear\\_model.LogisticRegression](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression)

## 3.2. 朴素贝叶斯

API:

```
sklearn.naive_bayes.GaussianNB(*, priors=None, var_smoothing=1e-09)
```

Reference:

[https://scikit-learn.org/stable/modules/generated/sklearn.naive\\_bayes.GaussianNB.html#sklearn.naive\\_bayes.GaussianNB](https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB)

### 3.3. K 近邻(KNN)

API:

```
sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None)
```

Reference:

<https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier>

### 3.4. 支持向量机(SVC)

API:

```
sklearn.svm.LinearSVC(penalty='l2', loss='squared_hinge', *, dual=True, tol=0.0001, C=1.0, multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0, random_state=None, max_iter=1000)
```

Reference:

<https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC>

### 3.5. 决策树

API:

```
sklearn.tree.DecisionTreeClassifier(*, criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0)
```

Reference:

<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier>

### 3.6. 随机森林

API:

```
sklearn.ensemble.RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None)
```

Reference:

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier>

## 3.7. 集成学习

### 3.7.1. AdaBoost

API:

```
sklearn.ensemble.RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None)
```

Reference:

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier>

### 3.7.2. GradientBoosting

API:

```
sklearn.ensemble.RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None)
```

Reference:

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier>

### 3.7.3. XGBoost

Docs: <https://xgboost.readthedocs.io/en/stable/python/index.html>

安装: `conda install -c conda-forge py-xgboost`

API:

```
xgboost.XGBClassifier(*, objective='binary:logistic', use_label_encoder=None, **kwargs)
```

### 3.7.4. LightGBM

Docs:

<https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMClassifier.html#lightgbm.LGBMClassifier>

安装: `pip install lightgbm`

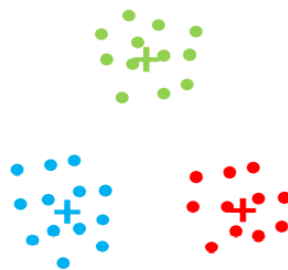
API:

```
lightgbm.LGBMClassifier(boosting_type='gbdt', num_leaves=31, max_depth=-1, learning_rate=0.1, n_estimators=100, subsample_for_bin=200000, objective=None, class_weight=None, min_split_gain=0.0, min_child_weight=0.001, min_child_samples=20, subsample=1.0, subsample_freq=0, colsample_bytree=1.0, reg_alpha=0.0, reg_lambda=0.0, random_state=None, n_jobs=None, importance_type='split', **kwargs)
```

## 4. 聚类算法

### 4.1. KMeans

**K-means** 算法是一种无监督学习方法，是最普及的聚类算法，算法使用一个没有标签的数据集，然后将数据聚类成不同的组。**K-means** 算法具有一个迭代过程，在这个过程中，数据集被分组成若干个预定义的不重叠的聚类或子组，使簇的内部点尽可能相似，同时试图保持簇在不同的空间，它将数据点分配给簇，以便簇的质心和数据点之间的平方距离之和最小，在这个位置，簇的质心是簇中数据点的算术平均值。



点集（簇）实例

#### **K-means** 算法流程

- 1、选择 **K** 个任意点作为初始质心。
- 2、将每个点指派到最近的质心，形成 **K** 个簇。
- 3、对于上一步聚类的结果，进行平均计算，得出该簇的新的聚类中心。
- 4、重复上述两步/直到迭代结束：质心不发生变化。

**K**-均值的一个问题在于，它有可能会停留在一个局部最小值处，而这取决于初始化的情况。为了解决这个问题，我们通常需要多次运行 **K**-均值算法，每一次都重新进行随机初始化，最后再比较多次运行 **K**-均值的结果，选择代价函数最小的结果。

#### **K-means** 的优点

- 1、原理比较简单，实现也是很容易，收敛速度快。
- 2、聚类效果较优。
- 3、算法的可解释度比较强。
- 4、主要需要调参的参数仅仅是簇数 **K**。

#### **K-means** 的缺点

- 1、需要预先指定簇的数量
- 2、如果有两个高度重叠的数据，那么它就不能被区分，也不能判断有两个簇。



- 3、有时随机选择质心并不能带来理想的结果。
- 4、无法处理异常值和噪声数据。
- 5、如果遇到非常大的数据集，那么计算机可能会崩溃。

API:

```
sklearn.cluster.KMeans(n_clusters=8, *, init='k-means++', n_init='warn', max_iter=300, tol=0.001, verbose=0, random_state=None, copy_x=True, algorithm='lloyd')
```

Reference:

<https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans>

## 4.2. 密度聚类 DBSCAN

**DBSCAN**(Density-Based Spatial Clustering of Applications with Noise)将簇定义为密度相连的点的最大集合，能够把具有足够高密度的区域划分为簇，并可在噪声的空间数据库中发现任意形状的聚类。

**密度**：空间中任意一点的密度是以该点为圆心，以**扫描半径**构成的圆区域内包含的点数目。

**扫描半径 (eps)**：用于定位点/检查任何点附近密度的距离度量，即扫描半径。

**最小包含点数(minPts)**：聚集在一起的最小点数（阈值），该区域被认为是稠密的。

DBSCAN 使用两个超参数（**扫描半径**和**最小包含点数**）来获得簇的数量，而不是猜测簇的数目。

DBSCAN 算法将数据点分为三类：

- 1、核心点：在半径 Eps 内含有超过 MinPts 数目的点。
- 2、边界点：在半径 Eps 内点的数量小于 MinPts，但是落在核心点的邻域内的点。
- 3、噪音点：既不是核心点也不是边界点的点。

DBSCAN 密度聚类的算法流程

- 1、将所有点标记为核心点、边界点或噪声点。
- 2、如果选择的点是核心点，则找出所有从该点出发的密度可达对象形成簇。
- 3、如果该点是非核心点，将其指派到一个与之关联的核心点的簇中。
- 4、重复以上步骤，直到所有点都被处理过。

API:

```
sklearn.cluster.DBSCAN(eps=0.5, *, min_samples=5, metric='euclidean', metric_params=None, algorithm='auto', leaf_size=30, p=None, n_jobs=None)
```

Reference:

<https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html#sklearn.cluster.DBSCAN>

## 5. 降维

**维数灾难(Curse of Dimensionality)**: 通常是指在涉及到向量的计算的问题中, 随着维数的增加, 计算量呈指数倍增长的一种现象。在很多机器学习问题中, 训练集中的每条数据经常伴随着上千、甚至上万个特征。要处理这所有的特征的话, 不仅会让训练非常缓慢, 还会极大增加搜寻良好解决方案的困难。

**降维(Dimensionality Reduction)**是将训练数据中的样本(实例)从高维空间转换到低维空间, 该过程与信息论中有损压缩概念密切相关, 不存在完全无损的降维。

作用:

- 1、减少冗余特征, 降低数据维度。
- 2、数据可视化, 例如: t-SNE、UMAP (后续案例)

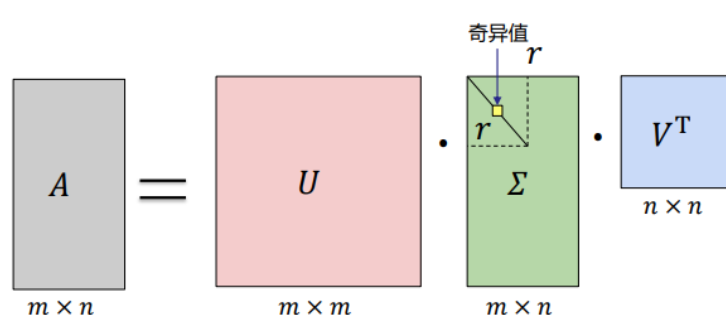
降维的优点:

- 1、通过减少特征的维数, 数据集存储所需的空间也相应减少, 减少了特征维数所需的计算训练时间;
- 2、数据集特征的降维有助于快速可视化数据;
- 3、通过处理多重共线性消除冗余特征。

降维的缺点:

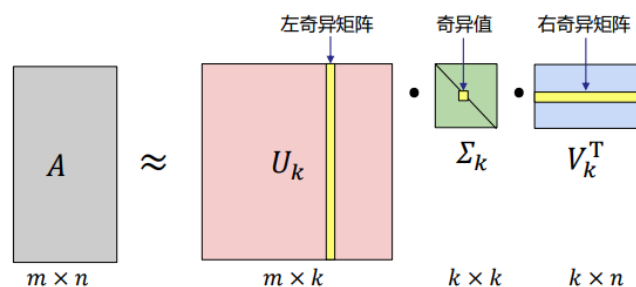
- 1、由于降维可能会丢失一些数据;
- 2、在主成分分析(PCA)降维技术中, 有时需要考虑多少主成分是难以确定的, 往往使用经验法则。

### 5.1. 奇异值分解 SVD

$$A = U \Sigma V^T$$


SVD 分解可以将一个矩阵进行分解, 对角矩阵对角线上的特征值递减存放, 而且奇异值的减少特别的快, 在很多情况下, 前 10%甚至 1%的奇异值的和就占了全部的奇异值之和的 99%以上的比例。也就是说, 对于奇异值, 它跟我们特征分解中的特征值类似, 我们也可以用最大的  $r (< \min(m, n))$  个的奇异值和对应的左右奇异向量来近似描述矩阵。

$$A_{m \times n} = U_{m \times m} \Sigma_{m \times n} V_{n \times n}^T \approx U_{m \times r} \Sigma_{r \times r} V_{r \times n}$$



API:

```
sklearn.decomposition.TruncatedSVD(n_components=2, *, algorithm='randomized', n_iter=5, n_oversamples=10, power_iteration_normalizer='auto', random_state=None, tol=0.0)
```

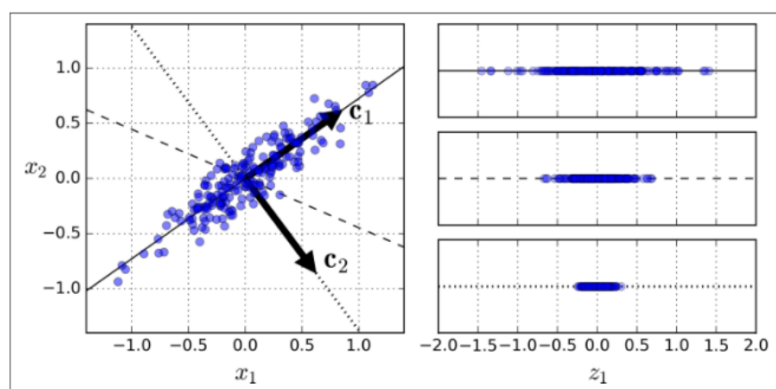
Reference:

<https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html#sklearn.decomposition.TruncatedSVD>

## 5.2. 主成分分析 PCA

**主成分分析**（Principal Component Analysis, PCA）通过将一个大的特征集转换成一个较小的特征集,这个特征集仍然包含了原始数据中的大部分信息,从而降低了原始数据的维数。减少一个数据集的特征数量自然是以牺牲准确性为代价的,用一点准确性换取简单性。因为更小的数据集更容易探索和可视化,并且对于机器学习算法来说,分析数据会更快、更容易,而不需要处理额外的特征。

PCA 确定各主成分的过程是通过平移、旋转坐标轴（维度），选择与前主成分正交且方差最大的轴。



通过计算数据矩阵的协方差矩阵，然后得到协方差矩阵的特征值特征向量，选择特征值最大(即方差最大)的  $k$  个特征所对应的特征向量组成的矩阵。这样就可以将数据矩阵转换到新的空间当中，实现数据特征的降维。

API:

```
sklearn.decomposition.PCA(n_components=None, *, copy=True, whiten=False, svd_solver='auto', tol=0.0, iterated_power='auto', n_oversamples=10, power_iteration_normalizer='auto', random_state=None)
```

Reference:

<https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html#sklearn.decomposition.PCA>

## 5.3. 非负矩阵分解 NMF

找到两个非负矩阵，即包含所有非负元素的矩阵( $W, H$ )，其乘积近似于非负矩阵  $x$ 。

$$X = WH$$

API:

```
sklearn.decomposition.NMF(n_components=None, *, init=None, solver='cd', beta_loss='frobenius', tol=0.0001, max_iter=200, random_state=None, alpha_W=0.0, alpha_H='same', l1_ratio=0.0, verbose=0, shuffle=False)
```

Reference:

<https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html#sklearn.decomposition.NMF>

## 6. 模型的评估方法和评价指标

### 6.1. 切分类

#### 6.1.1. K 折叠 KFold



- 1、数据集被分成  $K$  份（ $K$  通常取 5 或者 10）。
- 2、不重复地每次取其中一份做测试集，用其他  $K - 1$  份做训练集训练，这样会得到  $K$  个评价模型。
- 3、将上述步骤 2 中的  $K$  次评价的性能均值作为最后评价结果。

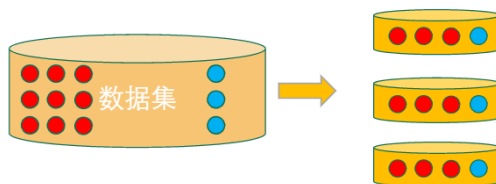
API:

```
sklearn.model_selection.KFold(n_splits=5, *, shuffle=False, random_state=None)
```

Reference:

[https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.KFold.html#sklearn.model\\_selection.KFold](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html#sklearn.model_selection.KFold)

### 6.1.2. 分层抽样策略 K 折叠



将数据集划分成  $k$  份，特点在于，划分的  $k$  份中，每一份内各个类别数据的比例和原始数据集中各个类别的比例相同。

API:

```
sklearn.model_selection.StratifiedKFold(n_splits=5, *, shuffle=False, random_state=None)
```

Reference:

[https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.StratifiedKFold.html#sklearn.model\\_selection.StratifiedKFold](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html#sklearn.model_selection.StratifiedKFold)

## 6.2. 切分函数

将样本集切分为训练集和测试集

API:

```
sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None)
```

Reference:

[https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.train\\_test\\_split.html#sklearn.model\\_selection.train\\_test\\_split](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html#sklearn.model_selection.train_test_split)

## 6.3. 超参数优化

对估计器的指定超参数值进行穷举搜索（网格搜索）。

API:

```
sklearn.model_selection.GridSearchCV(estimator, param_grid, *, scoring=None, n_jobs=None, refit=True, cv=None, verbose=0, pre_dispatch='2*n_jobs', error_score=nan, return_train_score=False)
```

Reference:

[https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.GridSearchCV.html#sklearn.model\\_selection.GridSearchCV](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV)

## 6.4. 模型验证

### 6.4.1. 交叉验证

计算交叉验证后的给定指标结果，评价指标详见：

[https://scikit-learn.org/stable/modules/model\\_evaluation.html#the-scoring-parameter-defining-model-evaluation-rules](https://scikit-learn.org/stable/modules/model_evaluation.html#the-scoring-parameter-defining-model-evaluation-rules)

API:

```
sklearn.model_selection.cross_validate(estimator, X, y=None, *, groups=None, scoring=None, cv=None, n_jobs=None, verbose=0, fit_params=None, pre_dispatch='2*n_jobs', return_train_score=False, return_estimator=False, error_score=nan)
```

Reference:

[https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.cross\\_validate.html#sklearn.model\\_selection.cross\\_validate](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html#sklearn.model_selection.cross_validate)

### 6.4.2. 交叉验证（分数）

返回一个 估计器做 k 折交叉验证后产生的 k 个在测试集上的 score（准确率）。返回值是一个有 k 个元素的数组。

API:

```
sklearn.model_selection.cross_val_score(estimator, X, y=None, *, groups=None, scoring=None, cv=None, n_jobs=None, verbose=0, fit_params=None, pre_dispatch='2*n_jobs', error_score=nan)
```

Reference:

[https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.cross\\_val\\_score.html#sklearn.model\\_selection.cross\\_val\\_score](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html#sklearn.model_selection.cross_val_score)

### 6.4.3. 交叉验证（预测值）

`cross_val_predict` 提供了和 `cross_val_score` 相似的接口，但是后者返回 k 次得分，而前者返回所有数据集上的预测结果。如果传入的参数 `method='predict'`（默认情况），那么返回 `(n_samples,)` 形状的 `ndarray`，如果传入的参数 `method='predict_proba'`，那么返回 `(n_samples, n_classes)` 形状的 `ndarray`。

API:

```
sklearn.model_selection.cross_val_predict(estimator, X, y=None, *, groups=None, cv=None, n_jobs=None, verbose=0, fit_params=None, pre_dispatch='2*n_jobs', method='predict')
```

Reference:

[https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.cross\\_val\\_predict.html#sklearn.model\\_selection.cross\\_val\\_predict](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_predict.html#sklearn.model_selection.cross_val_predict)

## 6.5. 分类、回归评价指标

`sklearn.metrics` 模块包含了一系列用于评价模型的评分函数、损失函数以及成对数据的距离度量函数。评价指标主要分为分类评价指标、回归评价指标等等，这里列举了常见的几种评价指标。

评价指标使用的方法如下：

```
from sklearn.metrics import 类名称
```

评价指标	库名称	使用范围
准确率	<code>accuracy_score</code>	分类
精确率	<code>precision_score</code>	分类
召回率	<code>recall_score</code>	分类
F1 值	<code>f1_score</code>	分类
对数损失	<code>log_loss</code>	分类
混淆矩阵	<code>confusion_matrix</code>	分类
Area Under the Receiver Operating Characteristic Curve (ROC AUC)	<code>roc_auc_score</code>	分类
平均精度 (mAP)	<code>average_precision_score</code>	分类
含多种评价的分类报告	<code>classification_report</code>	分类
均方误差 MSE	<code>mean_squared_error</code>	回归
平均绝对误差 MAE	<code>mean_absolute_error</code>	回归
决定系数 R2	<code>r2_score</code>	回归

参数：

**y\_true:** 真实标签

**y\_score:** 预测值 [0: 1]

**y\_pred:** 预测标签

Reference:

<https://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics>

<https://www.cnblogs.com/harvey888/p/6964741.html>

Official examples:

[https://scikit-learn.org/stable/modules/model\\_evaluation.html](https://scikit-learn.org/stable/modules/model_evaluation.html)



### 6.5.1. 混淆矩阵(confusion matrix)

混淆矩阵		真实值	
		0	1
预测值	0	TN	FP
	1	FN	TP

Negative 中文译作阴性，一般指标签 0；Positive 中文译作阳性，一般指标签 1。

True 中文译作预测正确；False 中文译作预测错误。

TN (True Negative): 预测正确 (True) 并且实际为阴性 (Negative) 即真实值和预测值均为 Negative。

TP (True Positive): 预测正确 (True) 并且实际为阳性 (Positive) 即真实值和预测值均为 Positive。

FN (False Negative): 预测错误 (False) 并且实际为阴性 (Negative) 即真实值为 Negative，预测值为 Positive。

FP (False Positive): 预测错误 (False) 并且实际为阳性 (Positive) 即真实值为 Positive，预测值为 Negative。

$$\text{accuracy(准确率)} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{precision(精确率)} = \frac{TP}{TP + FP}$$

$$\text{recall(召回率)} = \frac{TP}{TP + FN}$$

$$F_1 = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

例子：对于一个二分类问题，假设真实标签  $y\_labels=[1,1,0,1,1,0,0,0]$ ，我们预测的结果  $y\_scores=[0.8,0.9,0.6,0.3,0.7,0.1,0.1,0.6]$ 。假设  $threshold=0.5$ 。那么可以得到  $y\_preds=[1,1,1,0,1,0,0,1]$ 。这时我们可以得到混淆矩阵(confusion matrix)

混淆矩阵		真实值	
		0	1
预测值	0	2	1
	1	2	3

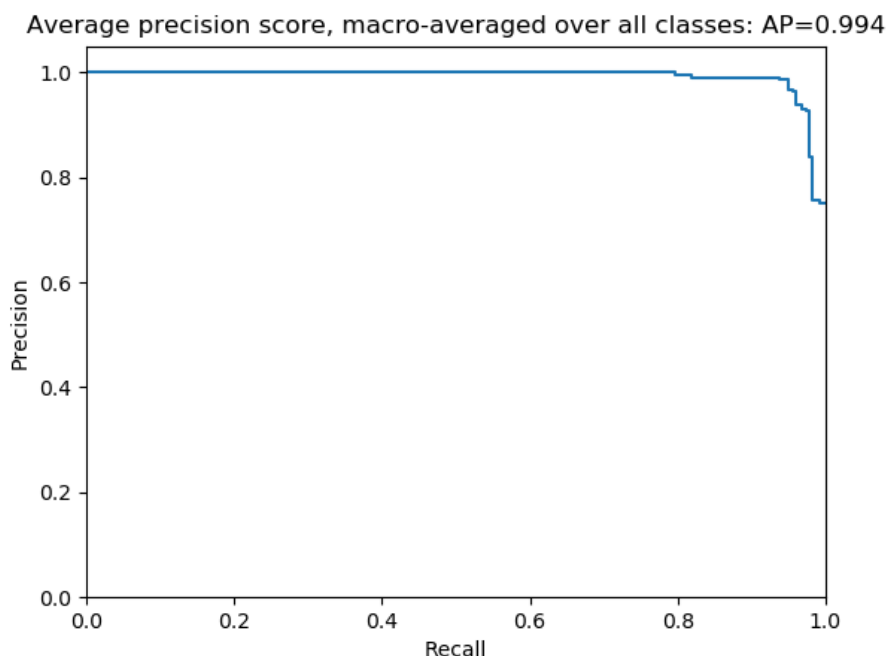
### 6.5.2. ROC/AUC/PR

ROC (Receiver Operating Characteristic) 曲线，又称接受者操作特征曲线。该曲线最早应用于雷达信号检测领域，用于区分信号与噪声。后来人们将其用于评价模型的预测能力，ROC 曲线是基于混淆矩阵得出的。

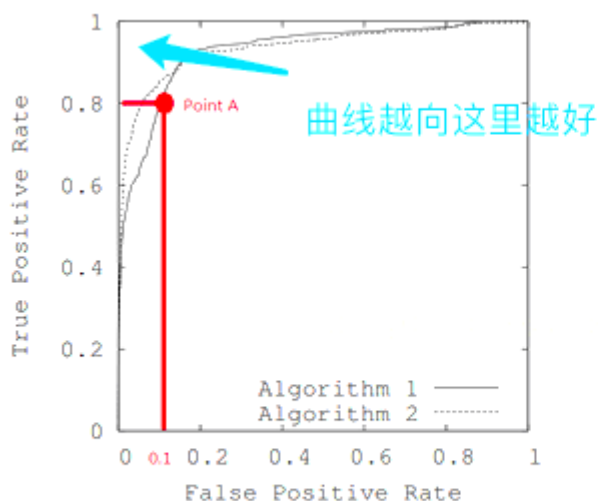


**AUPR:** 依次选择不同的正负阈值(或称为“截断点”),进而计算出不同的(Recall, Precision)对。画出全部的关键点以后,它们在坐标轴上对应了一条曲线,这条曲线就是 PR 曲线,曲线下的面积就是 AUPR 的值。

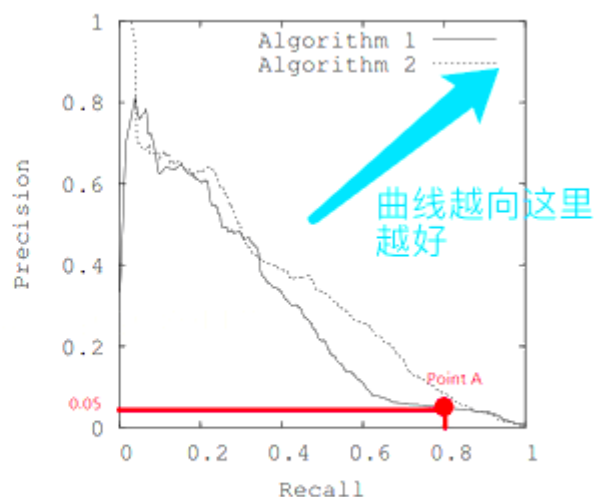
AUPR 和 mAP 是同一个含义,都是 precision-recall 曲线下的面积。只不过 AUPR 是折线,调用 `precision_recall_curve` 函数。mAP 是微积分的形式来近似曲线,调用 `average_precision_score` 函数。两者在值上差距很小。



**PR 和 ROC 区别:** 在面对不平衡数据时的表现是不同的。在数据不平衡时, PR 曲线是敏感的,随着正负样本比例的变化, PR 会发生强烈的变化。而 ROC 曲线是不敏感的,其曲线能够基本保持不变。



(a) Comparison in ROC space



(b) Comparison in PR space

### 6.5.3. micro 和 macro 区别

```
from sklearn.metrics import precision_score, recall_score, f1_score

a = [1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 3] #真实
b = [1, 1, 2, 1, 2, 2, 2, 2, 3, 1, 3, 3] #预测

print(precision_score(a, b, average='micro'), precision_score(a, b, average='macro'))
print(recall_score(a, b, average='micro'), recall_score(a, b, average='macro'))
print(f1_score(a, b, average='micro'), f1_score(a, b, average='macro'))

0.6666666666666666 0.7000000000000001
0.6666666666666666 0.6722222222222222
0.6666666666666666 0.6626984126984127
```

得出以下混淆矩阵结果：

	1 类	2 类	3 类	总数
TP	2	3	3	8
FP	2	2	0	4
FN	1	1	2	4
TN	7	6	7	20

micro：利用全局的 TP、FP 和 FN 进行计算

$$\text{micro - precision} = \frac{TP}{TP + FP} = \frac{8}{8 + 4} = 0.66666$$

$$\text{micro - recall} = \frac{TP}{TP + FN} = \frac{8}{8 + 4} = 0.66666$$

$$\text{micro} - F_1 = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}} = \frac{2 * 0.66 * 0.66}{0.66 + 0.66} = 0.666$$

macro：先计算各个类别的 TP、FP 和 FN，在进行均值汇总计算

$$\text{macro - precision} = \frac{1}{n} \sum_{i=1}^n \text{precision}_i = \frac{1}{3} \left( \frac{2}{2+2} + \frac{3}{3+2} + \frac{3}{3+0} \right) = \frac{7}{10}$$

$$\text{macro - recall} = \frac{1}{n} \sum_{i=1}^n \text{recall}_i = \frac{1}{3} \left( \frac{2}{2+1} + \frac{3}{3+1} + \frac{3}{3+2} \right) = \frac{121}{180} = 0.67222$$

$$\text{macro} - F_1 = \frac{1}{n} \sum_{i=1}^n F_{1i} = \frac{1}{n} \sum_{i=1}^n \frac{2 * \text{precision}_i * \text{recall}_i}{\text{precision}_i + \text{recall}_i} = 0.66269$$

## 7. 特征工程

sklearn.preprocessing 模块包含了数据变换的主要操作，如下表：

from sklearn.preprocessing import 类名称

预处理操作	类名称
标准化	StandardScaler
最小最大标准化	MinMaxScaler
One-Hot 编码	OneHotEncoder
归一化	Normalizer
二值化(单个特征转换)	Binarizer
标签编码	LabelEncoder
缺失值填补	Imputer
多项式特征生成	PolynomialFeatures

Reference:

<https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing>

Official examples:

<https://scikit-learn.org/stable/modules/preprocessing.html>

## 8. 机器学习药物发现案例（一）

——药物副作用预测模型（分类，矩阵分解算法）

## 9. 机器学习药物发现案例（二）

——基于随机森林的化合物活性二分类模型

## 10. 机器学习药物发现案例（三）

——基于随机森林和高斯回归的化合物活性预测（回归）